Queen: Browser-Based Distributed Computing Platform

CMPSC 450 Mini Project

Yuya Jeremy Ong

Outline

1. Introduction
2. Queen Platform
3. System Architecture
4. Development Process
5. Experiment
6. Discussion
7. Conclusion

Introduction

- High-Performance Computing often utilizes specialized hardware and are often build on a centralized monolithic architecture.
- Commodity hardware has become very ubiquitous!
  - Globally there are about ~4.1 billion devices.
  - Assume average device has 6.3 * 10 e 10 FLOPS (Intel i7 920 - 2.8 GHz)
  - **Global Computational Power = 0.27 ZFLOPS**
- Every computer has a browser, and all we do is watch cat videos and laugh at memes on the internet… wasted CPU cycles!

Challenge: Can we make use of the collective idle CPU cycles and allocate resources to people who need them most?

Queen: A Browser-Based Distributed Computing Platform

A client-server architecture which brokers a socketed communication between a **pool of browsers** to *perform computational tasks* on.

Key Features

1. Bidirectional Server-Client Socket Communication
2. JavaScript Development and 3rd Party Integration
3. System Identification using User-Agents or Modernizr
4. Automated connection and sandboxing using frameworks like Selenium.
5. Robust control mechanism for error handling and node failure cases.

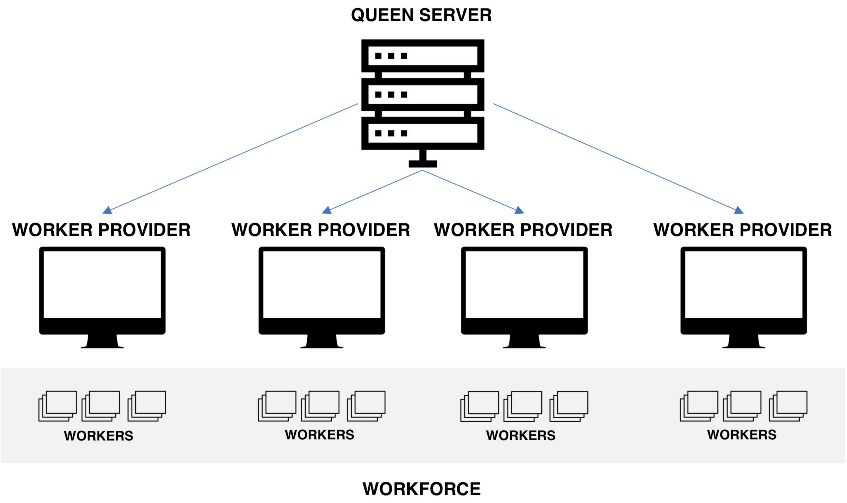
System Architecture

Queen Server: Server program which aggregates all collection of computers.

Workforce: The entire collection of worker providers that are connected to the Queen.

Worker Provider: Individual computer nodes which facilitates a browser connection to the Queen Server.

Workers: Individual iFrames that perform specific tasks administered by the Queen Server.



Development Process

Server Script

An initialization script responsible for the server side configurations and message handler.

```
// http://queenjs.com/ping-server.js
module.exports = function(queen){
	var config = {
		run: ['http://queenjs.com/ping-client.js'],

		// This tells queen to run this script on any
		// browsers which are connected now and in the future
		populate: "continuous",

		// By default, queen will kill a workforce (i.e. this job)
		// if there are no browsers connected, this tells queen
		// that it's ok to idle and wait for browsers to connect.
		killOnStop: false,

		// This function gets called right before a browser starts
		// running the client script.
		handler: function(worker){
			// Worker can be thought of as the browser.
			worker.on('message', function(num){
				queen.log(worker.provider + " is at " + num + "\n");

				// If the browser has pinged us 10 times, kill it.
				if(num === 10){
					worker.kill();
				} else {
					// Echo the number back to the worker
					worker(num);
				}
			});

			// Tell the worker to start at 0
			worker(0);
		}
	}

	// queen is a global variable of the running queen instance
	queen(config);
};
```

Worker Script

A message event handler that contains the script each node will be executing. It is injected during the initialization of the script in the iFrame.

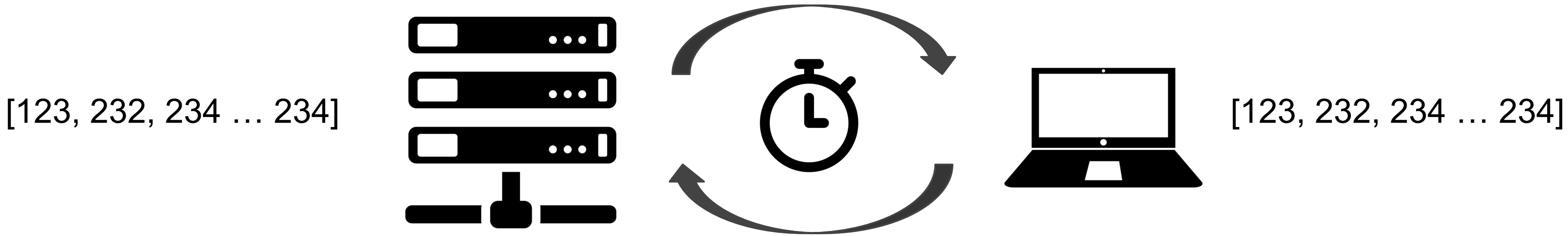
```
// http://queenjs.com/ping-client.js

// queenSocket is a global variable queen injects in
// to this context

// The queenSocket.onMessage hook executes whenever the server-side script
// sends a message.
queenSocket.onMessage = function(number){
	// Wait one second, then send the number + 1 back
	// to the server-side script
	setTimeout(function(){
		// Sending something in to the queenSocket function sends
		// it to the server-side script
		queenSocket(number + 1);
	}, 1000);
};
```

Experiment

Problem: ***Network Latency*** can cause a **huge bottleneck** in the *data transfer and instruction communication* process of the nodes.

Experiment Goal: Find out how much latency would such system experience?

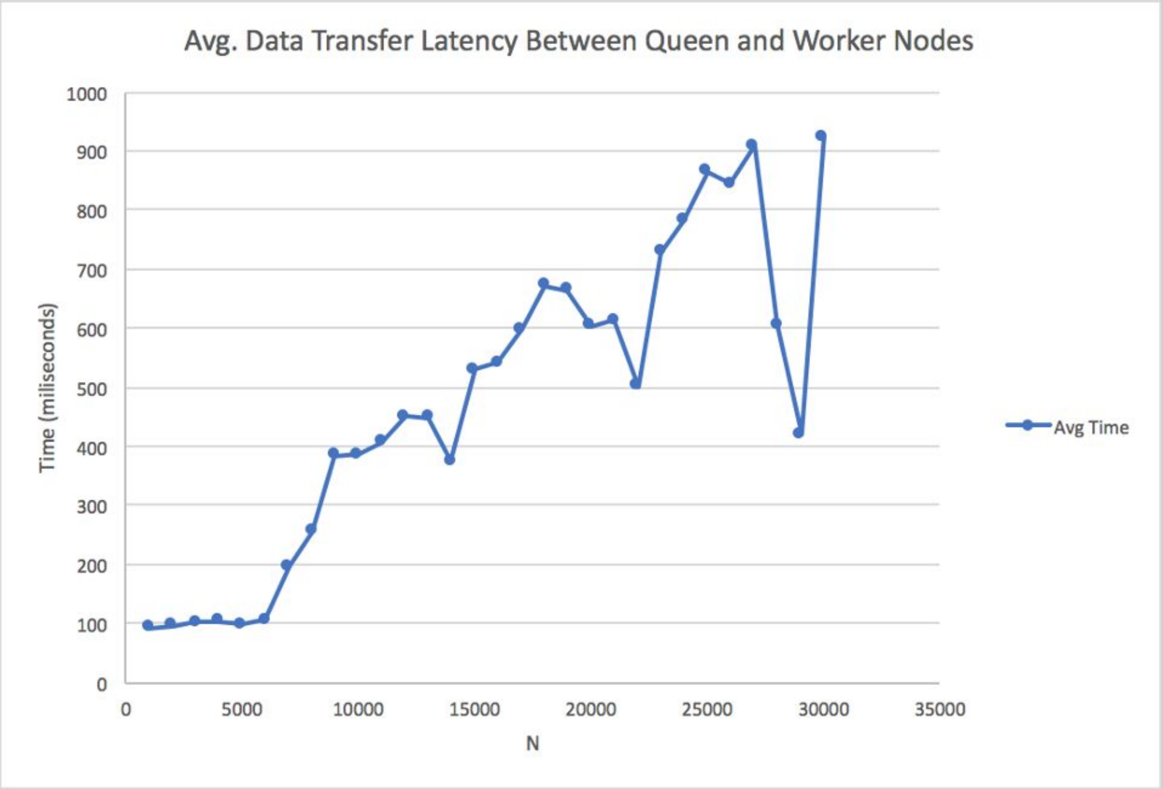
The Ping-Pong Latency Experiment



Environment Setup

- **Queen Server:** Deployed server application on aci-i node on port 9300 - exposed URL via secure tunneling utility called ngrok.
- **Client Code Server:** Client side script was hosted locally on a Macbook Pro via XAMPP server and exposed also through ngrok.
- **Worker Providers:**
    - Logged into 10 Workstation Mini-Towers at the Pattee Library Media Commons at 3 AM in the morning.
    - <u>Machine Specs:</u> Dell Optiplex 7050 Mini Tower
        - Intel Core i7-6700 @ 3.4 GHz
        - 16GB RAM
        - OS: Windows 10
        - Google Chrome Browsers

Experiment Results



Discussion

- Very novel and unorthodox architecture - has the potential for interesting applications and use cases if improved.
- Currently, far from being practical or useful in scientific applications.
- Network latency and large scale data transfer is a major problem.

<u>Potential Improvements to the System:</u>

- Use WebRTC Protocol (UDP Based) instead of WebSockets for ultra-low latency communication (cost of potential packet loss...).
- Make use of hardware based optimized libraries like OpenCL, GPU.JS, WASM (Web Assembly).
- Better orchestration and signaling between worker nodes would be interesting to implement.

Questions?